BOOK 1: WHAT IS PROGRAMMING
Chapter 1: The Computer Brain

WHAT COMPUTERS ACTUALLY DO

A computer is a machine that follows instructions. That is it. Nothing more.

Every video you watch, every game you play, every message you send is just a computer following instructions someone wrote. The computer does not think. The computer does not understand. The computer just does exactly what it is told.

Imagine you have a robot butler. This robot cannot think for itself. If you say "make me breakfast" it will stare at you because that instruction is too vague. But if you say "open the fridge, take out two eggs, crack them into the pan, turn on the stove to medium heat, wait three minutes, turn off the stove, put the eggs on a plate" then the robot can do it.

Computers are the same. They need exact, step by step instructions.

The computer brain is called a processor. It can do billions of simple operations per second. Addition. Subtraction. Comparing two numbers. Moving data from one place to another. These simple operations combine to create everything you see on your screen.

WHY WE NEED TO TALK TO COMPUTERS

Computers only understand one language: binary. Ones and zeros. On and off. Electricity flowing or not flowing.

Here is what "Hello" looks like in binary:

01001000 01100101 01101100 01101100 01101111

Nobody wants to write that. Nobody can read that easily. So humans invented programming languages. These languages sit between human language and binary. You write something a human can read, and the computer translates it to binary.

Think of it like having a translator. You speak English to the translator. The translator speaks binary to the computer. The computer does the work. The translator tells you what happened in English.

This translator is built into your computer. When you write code, the translator (called an interpreter or compiler) converts your words into ones and zeros the machine understands.

WHAT IS A PROGRAMMING LANGUAGE

A programming language is a set of rules for writing instructions that a computer can follow.

Just like English has grammar rules (subject before verb, periods at the end of sentences), programming languages have syntax rules. Break the rules and the computer will not understand you.

There are hundreds of programming languages. Python. Java. C. Ruby. Go. Rust. Each has different rules and different strengths.

Some languages are good for building websites. Some are good for building phone apps. Some are good for scientific calculations. Some are good for controlling robots.

You are going to learn JavaScript.

WHY JAVASCRIPT

JavaScript is special for several reasons.

First, it runs everywhere. Every web browser on every computer and phone in the world can run JavaScript. You do not need to install anything special. You already have everything you need.

Second, you see results immediately. Write some JavaScript, run it, see what happens. No waiting. No complex setup. This fast feedback helps you learn quickly.

Third, JavaScript is forgiving. Many languages will crash if you make small mistakes. JavaScript tries to keep going. This can cause problems later but for learning it means you spend less time fighting the language and more time understanding concepts.

Fourth, jobs. JavaScript is one of the most popular programming languages in the world. Companies need JavaScript developers. Learning JavaScript opens doors.

Fifth, JavaScript can do almost anything. Websites. Phone apps. Desktop apps. Servers. Games. Art. Music. Robots. If you can imagine it, JavaScript can probably help build it.

You are about to learn a skill that will change how you see the world. Every website you visit, every app you use, you will start to understand how it works. You will see the instructions behind the magic.

THE JOURNEY AHEAD

Learning to program is like learning to write. First you learn the alphabet. Then words. Then sentences. Then paragraphs. Then essays. Then books.

Right now you do not know the alphabet. That is fine. By the end of this book you will be writing sentences. By the end of this series you will be writing books.

Some chapters will feel easy. Some will feel hard. This is normal. The hard parts become easy with practice. The concepts that confuse you today will be obvious next month.

Do not skip ahead. Each chapter builds on the previous one. If you do not understand something, read it again. Try the examples. Break them. Fix them. This is how learning happens.

Your computer is waiting. It will do anything you tell it. You just need to learn how to speak its language.

The next chapter teaches you how to write your first instruction.

BOOK 1: WHAT IS PROGRAMMING
Chapter 2: Your First Words to the Computer

OPENING THE BROWSER CONSOLE

Every web browser has a secret tool built in. A place where you can write JavaScript and see it run immediately. This tool is called the console.

To open the console:

In Chrome: Press F12 then click Console tab
In Firefox: Press F12 then click Console tab
In Edge: Press F12 then click Console tab
In Safari: Press Command Option J

Or on any browser: Right click anywhere on a webpage, click Inspect, then click Console.

You should see a blank area with a blinking cursor. This is where you will write your first code.

The console is like a conversation with your computer. You type something, press Enter, and the computer responds. It will become your best friend while learning.

WRITING YOUR FIRST LINE

In the console, type exactly this:

console.log("Hello")

Now press Enter.

You should see the word Hello appear below what you typed.

Congratulations. You just wrote your first line of code. You gave an instruction to the computer and it followed that instruction.

Try it again with different words:

console.log("My name is a programmer now")

Press Enter. The computer displays your message.

Try with a number:

console.log(42)

The computer displays 42.

You are now a programmer. A beginner programmer, but a programmer. You wrote instructions. The computer followed them. That is programming.

WHAT JUST HAPPENED

Let us break down what you wrote:

console.log("Hello")

This has four parts.

console

This is an object that represents the console itself. Think of it as the name of the thing you are talking to. Like saying "Hey Console" before giving a command.

The dot .

The dot connects the object to the action. It says "Console, do this next thing."

log

This is the action. Log means "display this message." The word log comes from old ship captains who would write events in a logbook. In programming, logging means recording or displaying information.

The parentheses with "Hello" inside ("Hello")

The parentheses hold the information you want to display. Whatever you put inside the parentheses is what appears on screen. The quotes tell JavaScript that Hello is text, not a command.

Put together: "Hey Console, display the text Hello."

The computer does exactly that.

THE QUOTES MATTER

Try this in your console:

```
console.log(Hello)
```

Without the quotes. Press Enter.

You get an error. Something like "Hello is not defined."

Why? Without quotes, JavaScript thinks Hello is a variable name (a storage box we will learn about later). Since you never created a variable called Hello, JavaScript does not know what you mean.

The quotes tell JavaScript "this is just text, not a command or variable name."

You can use double quotes or single quotes:

```
console.log("Hello")
console.log('Hello')
```

Both work the same. Pick one style and stick with it. Most JavaScript programmers use single quotes or backticks (we will learn about backticks later).

THE SEMICOLON

You might see code written like this:

```
console.log("Hello");
```

With a semicolon at the end. The semicolon tells JavaScript "this instruction is complete."

JavaScript is smart enough to figure out where instructions end even without semicolons. Both of these work:

```
console.log("Hello")
console.log("Hello");
```

Some programmers always use semicolons. Some never do. This book will not use them because they are optional and removing them makes code cleaner. But if you see them in other code, now you know what they mean.

ERRORS ARE FRIENDS NOT ENEMIES

Type this in your console:

```
console.log("Hello
```

Press Enter. No closing quote.

You get an error. The console shows red text explaining what went wrong.

Errors are not failures. Errors are the computer telling you exactly what confused it. Read the error message. It says something about an unexpected end of input or unterminated string. This means you started a piece of text but never finished it.

Every programmer sees errors every day. Professional programmers with decades of experience see errors. The difference is they read the error, understand it, and fix it.

When you see an error:

1. Do not panic
2. Read what the error says
3. Look at your code
4. Find what is different from working code

5. Fix it
6. Try again

Errors teach you. Every error you fix makes you a better programmer. Embrace errors. They are showing you exactly where to look.

TRY THESE EXERCISES

In your console, try displaying:

Your name
Your age
Your favorite food
A sentence about your day
The number 12345
The number 3.14159

Each one should use console.log with the appropriate value inside.

If any of them give errors, read the error, figure out what went wrong, and fix it.

WHAT YOU LEARNED

You learned how to open the browser console.
You learned how to display text using console.log.
You learned that quotes mark text.
You learned that errors are helpful messages, not punishments.
You learned that you can talk to your computer and it will listen.

The next chapter teaches you how to store information so you can use it later.

BOOK 1: WHAT IS PROGRAMMING

Chapter 3: Storing Information

WHAT IS A VARIABLE

Imagine you have boxes. Each box has a label on the front. Inside each box you can put one thing. When you need that thing, you read the label, find the box, and take out what is inside.

Variables are boxes for information.

The label is the variable name. The thing inside is the value. You create a box, give it a name, put something in it, and later you can get that thing back by using the name.

In your console, type:

```
let myName = "Sarah"
```

Press Enter. Nothing visible happens but something important occurred. You created a box labeled myName and put the text Sarah inside it.

Now type:

```
console.log(myName)
```

Press Enter. The console displays Sarah.

You did not type "myName" with quotes. You typed myName without quotes. This tells JavaScript "go find the box labeled myName and give me what is inside."

The box remembers. Type it again:

```
console.log(myName)
```

Still Sarah. The box keeps holding the value until you change it or close the browser.

LET AND CONST

There are two main ways to create variables in JavaScript: let and const.

let creates a box you can change later:

```
let score = 0
console.log(score)

score = 10
console.log(score)

score = 25
console.log(score)
```

The value changes from 0 to 10 to 25. The let keyword allows this.

const creates a box you cannot change:

```
const birthYear = 1995
console.log(birthYear)

birthYear = 2000
```

That last line causes an error. "Assignment to constant variable." The const keyword means constant. Once set, it stays set.

When to use which:

Use const when the value should never change. A birth year does not change. A website URL does not change. A conversion rate (like

inches per centimeter) does not change.

Use let when the value might change. A score changes during a game. A user's current page changes as they navigate. A counter changes as you count.

When in doubt, start with const. If you later need to change the value, JavaScript will give you an error reminding you to switch to let.

NAMING RULES

Variable names have rules.

Names can contain letters, numbers, underscores, and dollar signs:

```
let userName = "Alex"
let user2 = "Blake"
let user_name = "Casey"
let $price = 9.99
```

Names cannot start with a number:

```
let 2ndPlace = "Silver"
```

This causes an error. Start with a letter, underscore, or dollar sign instead:

```
let secondPlace = "Silver"
```

Names cannot contain spaces:

```
let user name = "Alex"
```

This causes an error. Use camelCase instead (capitalize the first letter of each word except the first):

```
let userName = "Alex"
```

Names cannot be JavaScript reserved words. These are words JavaScript uses for its own purposes:

```
let let = 5
let const = 10
let if = 15
```

All errors. You cannot use words like let, const, if, else, for, while, function, return, and many others as variable names.

Names are case sensitive:

```
let name = "Alex"
let Name = "Blake"
let NAME = "Casey"
```

These are three different variables. Lowercase name, capitalized Name, and uppercase NAME are all separate boxes.

NAMING CONVENTIONS

The rules tell you what JavaScript allows. Conventions tell you what programmers expect.

Use descriptive names:

```
let x = 5
```

What is x? Nobody knows. Instead:

```
let playerAge = 5
let numberOfLives = 5
let secondsRemaining = 5
```

Anyone reading this code understands what each variable holds.

Use camelCase for variable names. Start lowercase, capitalize each new word:

```
let firstName = "Alex"
let lastName = "Smith"
let dateOfBirth = "1995-03-15"
let isLoggedIn = true
```

Do not use single letters except for simple counters:

```
let i = 0
```

The letter i is traditionally used for counting in loops. You will learn this later. For everything else, use full words.

Do not abbreviate unless the abbreviation is obvious:

```
let usrNm = "Alex"
```

Confusing. Instead:

```
let userName = "Alex"
```

Clear.

CHANGING WHAT IS INSIDE THE BOX

With let variables, you can change the value:

```
let mood = "happy"
console.log(mood)

mood = "excited"
console.log(mood)

mood = "tired"
```

```
console.log(mood)
```

Notice you only use let once, when creating the variable. After that, just use the name:

```
let mood = "happy"
let mood = "excited"
```

This causes an error. "Identifier mood has already been declared." You cannot create two boxes with the same name. Create once, then just change the contents.

You can change a variable based on its current value:

```
let count = 0
console.log(count)

count = count + 1
console.log(count)

count = count + 1
console.log(count)
```

The line count = count + 1 means "take what is in count, add 1 to it, put the result back in count."

Start with 0. Add 1, now it is 1. Add 1 again, now it is 2.

There is a shortcut for this common operation:

```
let count = 0
count++
console.log(count)

count++
console.log(count)
```

The ++ means "add 1 to this variable." It is called the increment operator.

There is also -- for subtracting 1:

```
let lives = 3
lives--
console.log(lives)

lives--
console.log(lives)
```

Lives goes from 3 to 2 to 1.

PRACTICE

Create variables for:

Your first name (const because it does not change)
Your current age (let because it changes yearly)
Your favorite color (const if you are certain, let if you might change your mind)
A counter starting at 0 (let because counters change)

Display each one using console.log.

Change the counter a few times using ++ and display it after each change.

WHAT YOU LEARNED

You learned that variables are boxes with names that hold values.
You learned that let creates changeable variables.
You learned that const creates unchangeable variables.
You learned the rules for naming variables.
You learned conventions that make code readable.
You learned how to change variable values.

The next chapter explains the different types of information you can store.

BOOK 1: WHAT IS PROGRAMMING

Chapter 4: Types of Information

DIFFERENT KINDS OF DATA

Not all information is the same. A person's name is different from their age. A price is different from whether something is available. JavaScript recognizes these differences and treats each type of data appropriately.

The main types you will use are:

Numbers (for math and counting)
Strings (for text)
Booleans (for yes/no, true/false)
Null (for intentional emptiness)
Undefined (for accidental emptiness)

Understanding types helps you avoid bugs and write code that makes sense.

NUMBERS

Numbers in JavaScript work like numbers in math. You can use whole numbers and decimals.

```
let age = 25
let temperature = 98.6
```

```
let price = 19.99
let negative = -50
let zero = 0
```

No quotes around numbers. If you put quotes, JavaScript thinks it is text:

```
let realNumber = 42
let textNumber = "42"
```

These are different. The first is a number you can do math with. The second is text that happens to look like a number.

You can check what type something is:

```
console.log(typeof realNumber)
console.log(typeof textNumber)
```

The first shows "number". The second shows "string". The typeof operator tells you what type of data you have.

JavaScript handles very large and very small numbers:

```
let big = 999999999999
let tiny = 0.000000001
```

There are limits but you will rarely hit them in normal programming.

There is a special number called Infinity:

```
let huge = Infinity
console.log(huge)
console.log(huge + 1)
```

Infinity plus 1 is still Infinity. This shows up when you divide by zero or calculate something too large.

There is also NaN which means "Not a Number":

```
let weird = "hello" / 2
console.log(weird)
```

You cannot divide text by 2. The result is NaN. This tells you something went wrong with a math operation.

STRINGS

Strings are text. The name comes from "a string of characters" like beads on a string.

Create strings with quotes:

```
let singleQuotes = 'Hello World'
let doubleQuotes = "Hello World"
let backticks = `Hello World`
```

All three work. Backticks have special powers we will learn later.

Strings can contain anything:

```
let sentence = "The quick brown fox jumps over the lazy dog."
let withNumbers = "I have 3 cats and 2 dogs."
let withSymbols = "Email: test@example.com"
let empty = ""
```

An empty string has nothing between the quotes. It is still a string, just with zero characters.

What if your text contains quotes?

```
let problem = "She said "Hello" to me"
```

This causes an error. JavaScript sees the quote before Hello and thinks the string ended.

Solutions:

Use different quotes outside:

```
let solution1 = 'She said "Hello" to me'
let solution2 = "She said 'Hello' to me"
```

Single quotes outside let you use double quotes inside and vice versa.

Or use backslash to escape:

```
let solution3 = "She said \"Hello\" to me"
```

The backslash tells JavaScript "the next character is just text, not a command."

Strings have a length:

```
let greeting = "Hello"
console.log(greeting.length)
```

This shows 5. The word Hello has 5 characters.

BOOLEANS

Booleans have only two possible values: true or false.

```
let isRaining = true
let isSunny = false
let hasAccount = true
let isLoggedIn = false
```

No quotes around true and false. They are special keywords.

Booleans answer yes/no questions:

Is the user logged in? true or false.

Is the password correct? true or false.
Has the file finished downloading? true or false.
Is the number greater than 10? true or false.

The name boolean comes from George Boole, a mathematician who developed the logic that computers use.

You will use booleans constantly when making decisions in code. The next book covers this in depth.

Check the type:

```
console.log(typeof true)
console.log(typeof false)
```

Both show "boolean".

NULL

Null means "intentionally empty." You use it when you want to say "this variable exists but has no value on purpose."

```
let selectedUser = null
```

This might mean no user is currently selected. The variable exists and has a value (null), but that value represents "nothing chosen yet."

Null is different from not having a variable at all. It is a deliberate choice to represent emptiness.

Check the type:

```
console.log(typeof null)
```

This shows "object" which is actually a bug in JavaScript from the 1990s that was never fixed. It should show "null" but it does not.

Just remember that null is its own type.

UNDEFINED

Undefined means "this exists but nobody gave it a value."

```
let mystery
console.log(mystery)
```

We created a variable but never put anything in it. JavaScript automatically sets it to undefined.

Undefined often indicates a mistake. You meant to set a value but forgot. Or you tried to access something that does not exist.

```
let person = { name: "Alex" }
console.log(person.age)
```

This shows undefined. The person object has a name but no age. Asking for something that does not exist gives undefined.

Check the type:

```
console.log(typeof undefined)
```

This shows "undefined" correctly.

NULL VS UNDEFINED

Both represent "nothing" but mean different things.

Null: "I deliberately set this to nothing"
Undefined: "Nobody ever set this to anything"

let userChoice = null

Meaning: The user has not chosen yet, but we are tracking their choice.

let userChoice

Meaning: We created a variable but forgot to use it.

In practice, use null when you want to clear something or represent "no value chosen." Let JavaScript use undefined for things you forgot.

WHY TYPES MATTER

Different types behave differently:

```
console.log(5 + 3)
console.log("5" + "3")
```

The first shows 8. Addition of numbers.
The second shows 53. Joining of strings.

The plus sign does different things depending on what types you give it. With numbers, it adds. With strings, it joins.

This can cause bugs:

```
let price = "10"
let tax = 2
let total = price + tax
console.log(total)
```

You might expect 12. You get "102". The string "10" joined with the number 2, which JavaScript converted to "2", giving "102".

Always know what types your variables hold. When something

strange happens, check the types.

PRACTICE

Create these variables with appropriate types:

Your age (number)
Your name (string)
Whether you like pizza (boolean)
Your middle name if you have one, null if you do not
A variable with no value assigned

Use typeof on each one to verify the types.

Try adding a number and a string. See what happens.

WHAT YOU LEARNED

You learned that JavaScript has different types of data.
You learned that numbers are for math and have no quotes.
You learned that strings are for text and need quotes.
You learned that booleans are true or false for yes/no logic.
You learned that null means intentionally empty.
You learned that undefined means accidentally empty.
You learned that types affect how operations work.

The next chapter teaches you how to do math with numbers.

ADDITION SUBTRACTION MULTIPLICATION DIVISION

JavaScript can do math. The symbols are almost what you expect from school.

Addition uses the plus sign:

```
console.log(5 + 3)
```

Shows 8.

Subtraction uses the minus sign:

```
console.log(10 - 4)
```

Shows 6.

Multiplication uses the asterisk (the star symbol):

```
console.log(6 * 7)
```

Shows 42. We use asterisk instead of x because x looks like a variable name.

Division uses the forward slash:

```
console.log(20 / 4)
```

Shows 5.

You can use decimals:

```
console.log(10.5 + 2.3)
console.log(7.5 / 2.5)
```

Shows 12.8 and 3.

You can use negative numbers:

```
console.log(-5 + 3)
console.log(10 - -3)
```

Shows -2 and 13. Subtracting a negative is the same as adding.

REMAINDER (MODULO)

There is an operator you might not know from school. The percent sign gives you the remainder after division.

```
console.log(10 % 3)
```

10 divided by 3 is 3 with remainder 1. This shows 1.

```
console.log(17 % 5)
```

17 divided by 5 is 3 with remainder 2. This shows 2.

```
console.log(20 % 4)
```

20 divided by 4 is 5 with remainder 0. This shows 0.

Why is this useful? Common uses:

Checking if a number is even or odd:

```
console.log(6 % 2)
console.log(7 % 2)
```

Even numbers give 0 when divided by 2. Odd numbers give 1.

Wrapping around. If you have 12 hours on a clock and add 5 hours to 10:

```
console.log((10 + 5) % 12)
```

Shows 3. After 12 it wraps back to the start.

Cycling through a list. If you have 4 items and keep moving forward:

```
let position = 0
position = (position + 1) % 4
position = (position + 1) % 4
position = (position + 1) % 4
position = (position + 1) % 4
position = (position + 1) % 4
```

The position goes 0, 1, 2, 3, 0, 1, 2, 3 forever. It never goes above 3.

EXPONENTS

To raise a number to a power, use two asterisks:

```
console.log(2 ** 3)
```

2 to the power of 3. That is 2 times 2 times 2. Shows 8.

```
console.log(10 ** 2)
```

10 squared. Shows 100.

```
console.log(5 ** 0)
```

Any number to the power of 0 is 1. Shows 1.

You can do square roots with the power of 0.5:

```
console.log(16 ** 0.5)
```

Square root of 16. Shows 4.

```
console.log(27 ** (1/3))
```

Cube root of 27. Shows 3.

ORDER OF OPERATIONS

JavaScript follows the same order of operations you learned in school. PEMDAS or BODMAS depending on where you went to school.

Parentheses first
Exponents second
Multiplication and Division third (left to right)
Addition and Subtraction fourth (left to right)

console.log(2 + 3 * 4)

Multiplication happens before addition. 3 times 4 is 12. Then 2 plus 12 is 14. Shows 14.

console.log((2 + 3) * 4)

Parentheses first. 2 plus 3 is 5. Then 5 times 4 is 20. Shows 20.

console.log(10 - 4 - 2)

Left to right for same priority. 10 minus 4 is 6. Then 6 minus 2 is 4. Shows 4.

console.log(10 / 2 * 5)

Left to right. 10 divided by 2 is 5. Then 5 times 5 is 25. Shows 25.

When in doubt, use parentheses to make your intention clear:

let result = ((price * quantity) + shipping) * taxRate

Even if some parentheses are technically unnecessary, they make

the code easier to read.

COMBINING MATH WITH VARIABLES

You rarely do math with raw numbers. Usually you use variables:

```
let price = 29.99
let quantity = 3
let subtotal = price * quantity
console.log(subtotal)
```

Shows 89.97.

```
let totalSeconds = 3725
let minutes = Math.floor(totalSeconds / 60)
let seconds = totalSeconds % 60
console.log(minutes)
console.log(seconds)
```

This converts seconds to minutes and remaining seconds. 3725 seconds is 62 minutes and 5 seconds.

Math.floor rounds down to the nearest whole number. We will learn more about the Math object later.

```
let fahrenheit = 98.6
let celsius = (fahrenheit - 32) * 5 / 9
console.log(celsius)
```

Temperature conversion. Shows 37 (approximately).

```
let radius = 5
let area = 3.14159 * radius ** 2
console.log(area)
```

Area of a circle. Shows approximately 78.54.

SHORTHAND OPERATORS

There are shortcuts for common math operations on variables.

Instead of:

```
let score = 100
score = score + 10
```

You can write:

```
let score = 100
score += 10
```

The += means "add this to the existing value."

All math operations have shortcuts:

```
let num = 20

num += 5
console.log(num)

num -= 3
console.log(num)

num *= 2
console.log(num)

num /= 4
console.log(num)
```

Starting with 20:
Add 5, now 25
Subtract 3, now 22
Multiply by 2, now 44
Divide by 4, now 11

These shortcuts make code shorter and clearer when you are updating a variable based on its current value.

INCREMENT AND DECREMENT

The most common operation is adding or subtracting 1. JavaScript has special shortcuts:

```
let count = 0

count++
console.log(count)

count++
console.log(count)

count--
console.log(count)
```

Starting with 0:
Increment, now 1
Increment, now 2
Decrement, now 1

These are used constantly in loops (coming in Book 3).

COMMON MISTAKES

Dividing by zero:

```
console.log(10 / 0)
```

Shows Infinity. This might not be what you wanted.

Floating point weirdness:

console.log(0.1 + 0.2)

Shows 0.30000000000000004 instead of 0.3. This is because computers store decimals in binary, which cannot perfectly represent some numbers. For most purposes this tiny error does not matter. For money, you should store cents as whole numbers.

Forgetting order of operations:

let average = 80 + 90 + 85 / 3

You wanted the average of three numbers. But division happens first. You get 80 + 90 + 28.33 = 198.33 instead of 85.

Correct:

let average = (80 + 90 + 85) / 3

Shows 85.

PRACTICE

Calculate these in your console:

The sum of 15, 27, and 42
The product of 8 and 13
125 divided by 5
The remainder when 100 is divided by 7
3 to the power of 4
The average of 72, 85, 91, and 68

Create a variable for a price (like 49.99), a variable for quantity (like 3), and calculate the total.

Create a variable with 1000 seconds. Convert it to minutes and remaining seconds.

WHAT YOU LEARNED

You learned that JavaScript does math with + - * / symbols.
You learned that percent gives the remainder (modulo).
You learned that double asterisk does exponents.
You learned that order of operations follows math rules.
You learned shorthand operators like += and ++.
You learned to watch out for floating point errors.

The next chapter teaches you how to work with text.
BOOK 1: WHAT IS PROGRAMMING
Chapter 6: Working with Text

JOINING TEXT TOGETHER

When you use the plus sign with strings, it joins them. This is called concatenation.

```
let firstName = "Sarah"
let lastName = "Johnson"
let fullName = firstName + lastName
console.log(fullName)
```

Shows SarahJohnson. No space between them because we did not add one.

```
let fullName = firstName + " " + lastName
console.log(fullName)
```

Shows Sarah Johnson. We added a space string in the middle.

You can concatenate as many strings as you want:

```
let greeting = "Hello" + " " + "there" + " " + "friend"
console.log(greeting)
```

Shows Hello there friend.

You can mix variables and literal strings:

```
let name = "Alex"
let message = "Welcome, " + name + "!"
console.log(message)
```

Shows Welcome, Alex!

Numbers get converted to strings when concatenated:

```
let age = 25
let message = "I am " + age + " years old"
console.log(message)
```

Shows I am 25 years old. The number 25 became the text "25" automatically.

This automatic conversion can cause surprises:

```
let result = "The answer is: " + 5 + 3
console.log(result)
```

Shows The answer is: 53 not The answer is: 8. JavaScript converted 5 to "5", concatenated with 3 (converted to "3"), giving "53".

To do math first, use parentheses:

```
let result = "The answer is: " + (5 + 3)
console.log(result)
```

Shows The answer is: 8. Parentheses force the addition before concatenation.

TEMPLATE LITERALS

Template literals are a modern way to build strings. Use backticks instead of quotes.

```
let name = "Alex"
let greeting = `Hello, ${name}!`
console.log(greeting)
```

Shows Hello, Alex!

The backtick is usually on the same key as tilde, top left of your keyboard.

The dollar sign and curly braces ${} create a placeholder. JavaScript evaluates what is inside and inserts the result.

You can put any expression inside:

```
let price = 19.99
let quantity = 3
let message = `Total: $${price * quantity}`
console.log(message)
```

Shows Total: $59.97. The math happens inside the placeholder.

```
let a = 10
let b = 5
let result = `${a} plus ${b} equals ${a + b}`
console.log(result)
```

Shows 10 plus 5 equals 15.

Template literals can span multiple lines:

```
let poem = `Roses are red
Violets are blue
JavaScript is fun
And so are you`
console.log(poem)
```

This prints on four lines. Regular strings cannot do this without special characters.

Template literals make complex strings much easier to read:

Concatenation way:

```
let html = "<div class=\"user\">" + "<h2>" + userName + "</h2>" + "<p>" + userBio + "</p>" + "</div>"
```

Template literal way:

```
let html = `<div class="user">
 <h2>${userName}</h2>
 <p>${userBio}</p>
</div>`
```

The second version is much clearer. You can see the structure.

FINDING LENGTH

Every string knows how many characters it contains.

```
let word = "Hello"
console.log(word.length)
```

Shows 5. Five characters: H, e, l, l, o.

```
let sentence = "The quick brown fox"
console.log(sentence.length)
```

Shows 19. Spaces count as characters.

```
let empty = ""
console.log(empty.length)
```

Shows 0. An empty string has zero characters.

Length is useful for validation:

```
let password = "abc123"
console.log(password.length)
```

If password.length is less than 8, you might reject it as too short.

```
let tweet = "This is my message to the world!"
console.log(tweet.length)
```

If tweet.length is more than 280, it is too long for Twitter.

ACCESSING LETTERS BY POSITION

Each character in a string has a position called an index. Positions start at 0, not 1.

```
let word = "Hello"
```

Position 0: H
Position 1: e
Position 2: l
Position 3: l
Position 4: o

To get a character at a position, use square brackets:

```
let word = "Hello"
console.log(word[0])
```

```
console.log(word[1])
console.log(word[4])
```

Shows H, e, o.

The last character is at position length minus 1:

```
let word = "Hello"
let lastIndex = word.length - 1
console.log(word[lastIndex])
```

Shows o. The word has length 5, so the last index is 4.

Asking for a position that does not exist gives undefined:

```
let word = "Hello"
console.log(word[10])
```

Shows undefined. There is no character at position 10.

You can also use the at() method:

```
let word = "Hello"
console.log(word.at(0))
console.log(word.at(-1))
```

The at() method accepts negative numbers. -1 means the last character, -2 means second to last, and so on.

```
console.log(word.at(-1))
console.log(word.at(-2))
```

Shows o and l.

STRINGS ARE IMMUTABLE

You cannot change a character in an existing string:

```
let word = "Hello"
word[0] = "J"
console.log(word)
```

Still shows Hello, not Jello. The assignment silently failed.

To change a string, you must create a new string:

```
let word = "Hello"
word = "J" + word.slice(1)
console.log(word)
```

Shows Jello. We created a new string starting with J and added everything from position 1 onward.

The slice method extracts part of a string. We will learn more string methods later.

ESCAPE CHARACTERS

Some characters are hard to type directly. Escape sequences let you include them.

Newline (start a new line):

```
let twoLines = "First line\nSecond line"
console.log(twoLines)
```

Tab (indentation):

```
let indented = "Name:\tAlex"
console.log(indented)
```

Backslash (since backslash is the escape character):

```
let path = "C:\\Users\\Documents"
```

```
console.log(path)
```

Quote inside a string:

```
let quote = "She said \"Hello\""
console.log(quote)
```

With template literals, you rarely need escape sequences because you can use the other quote type or multiple lines naturally.

PRACTICE

Create variables for your first name and last name. Concatenate them with a space between.

Create the same full name using a template literal.

Find the length of your full name.

Get the first character of your name and the last character.

Create a multiline string with your address using a template literal.

Create a string that includes a quote character.

WHAT YOU LEARNED

You learned that plus joins strings together (concatenation).
You learned that template literals use backticks and ${} placeholders.
You learned that strings have a length property.
You learned that characters are accessed by index starting at 0.
You learned that strings cannot be changed, only replaced.
You learned escape sequences for special characters.

The next chapter teaches you how to compare things.

BOOK 1: WHAT IS PROGRAMMING

Chapter 7: Comparing Things

EQUAL AND NOT EQUAL

Comparisons ask questions and get true or false answers.

Is 5 equal to 5?

```
console.log(5 === 5)
```

Shows true.

Is 5 equal to 3?

```
console.log(5 === 3)
```

Shows false.

The triple equals === checks if two things are the same.

Is this string equal to that string?

```
console.log("hello" === "hello")
console.log("hello" === "Hello")
```

First shows true. Second shows false. Capitalization matters.

To check if things are NOT equal, use !==

```
console.log(5 !== 3)
```

Shows true. 5 is not equal to 3.

```
console.log(5 !== 5)
```

Shows false. 5 is equal to 5, so "not equal" is false.

You can compare variables:

```
let age = 25
let requiredAge = 18

console.log(age === requiredAge)
console.log(age !== requiredAge)
```

First shows false (25 is not 18). Second shows true (25 is not 18).

GREATER AND LESS

Is 10 greater than 5?

```
console.log(10 > 5)
```

Shows true.

Is 3 greater than 7?

```
console.log(3 > 7)
```

Shows false.

Is 4 less than 9?

```
console.log(4 < 9)
```

Shows true.

Is 8 less than 2?

```
console.log(8 < 2)
```

Shows false.

Greater than or equal uses > =

```
console.log(5 > = 5)
console.log(5 > = 4)
console.log(5 > = 6)
```

First two show true. Last shows false.

Less than or equal uses < =

```
console.log(5 < = 5)
console.log(5 < = 6)
console.log(5 < = 4)
```

First two show true. Last shows false.

You can compare strings. JavaScript compares alphabetically:

```
console.log("apple" < "banana")
```

Shows true. Apple comes before banana alphabetically.

```
console.log("cat" > "car")
```

Shows true. When the first two letters are the same, it compares the third. T comes after R.

Uppercase letters come before lowercase in JavaScript's ordering:

```
console.log("Z" < "a")
```

Shows true. This surprises many people. All uppercase letters have lower values than all lowercase letters.

STRICT VS LOOSE EQUALITY

We have been using `===` which is strict equality. There is also `==` which is loose equality.

```
console.log(5 === "5")
console.log(5 == "5")
```

First shows false. The number 5 is not the same as the string "5". Different types.
Second shows true. Loose equality converts types before comparing.

```
console.log(0 === false)
console.log(0 == false)
```

First shows false. Zero is a number, false is a boolean.
Second shows true. Loose equality considers 0 and false equivalent.

```
console.log(null === undefined)
console.log(null == undefined)
```

First shows false. Different types.
Second shows true. Loose equality considers them equivalent.

Always use strict equality (`===` and `!==`). Loose equality causes bugs because it does unexpected conversions. You think you are comparing numbers but one is actually a string and loose equality hides the problem.

The only time loose equality is sometimes used:

```
if (value == null)
```

This checks if value is either null or undefined. But you can write it explicitly instead:

```
if (value === null || value === undefined)
```

Stick to strict equality. Your code will be more predictable.

COMBINING COMPARISONS

Often you need to check multiple conditions.

AND (both must be true):

```
let age = 25
let hasLicense = true

console.log(age >= 18 && hasLicense === true)
```

Shows true. Both conditions are true.

The && symbol means AND. Both sides must be true for the result to be true.

```
console.log(true && true)
console.log(true && false)
console.log(false && true)
console.log(false && false)
```

Only the first shows true. All others show false.

OR (at least one must be true):

```
let isWeekend = false
let isHoliday = true

console.log(isWeekend || isHoliday)
```

Shows true. At least one is true.

The || symbol means OR. If either side is true, the result is true.

```
console.log(true || true)
```

```
console.log(true || false)
console.log(false || true)
console.log(false || false)
```

Only the last shows false. All others show true.

NOT (flips true to false and false to true):

```
let isRaining = true
console.log(!isRaining)
```

Shows false. The exclamation mark flips the value.

```
console.log(!true)
console.log(!false)
```

Shows false then true.

You can combine these:

```
let age = 25
let hasPermission = false
let isAdmin = true

console.log(age >= 18 && (hasPermission || isAdmin))
```

Shows true. Age is over 18 AND (hasPermission OR isAdmin is true).

Use parentheses to make complex logic clear.

THE RESULT IS A BOOLEAN

All comparisons produce booleans. You can store them:

```
let age = 25
let isAdult = age >= 18
```

```
console.log(isAdult)
```

Shows true. The variable isAdult holds the result of the comparison.

```
let password = "abc123"
let isLongEnough = password.length >= 8
console.log(isLongEnough)
```

Shows false. The password is only 6 characters.

This is useful for making decisions (covered in Book 2):

```
let canVote = age >= 18 && isCitizen
let canDrive = age >= 16 && hasLicense
let needsParent = age < 18
```

COMPARING OBJECTS AND ARRAYS

Objects and arrays compare differently. They compare by reference, not by content.

```
let arr1 = [1, 2, 3]
let arr2 = [1, 2, 3]
console.log(arr1 === arr2)
```

Shows false. Even though they contain the same values, they are two different arrays in memory.

```
let arr1 = [1, 2, 3]
let arr2 = arr1
console.log(arr1 === arr2)
```

Shows true. Both variables point to the same array.

We will learn more about this when we cover arrays and objects in depth.

PRACTICE

Check if 10 is greater than 5.
Check if "hello" equals "hello".
Check if 100 is less than or equal to 100.
Check if the number 7 strictly equals the string "7".
Check if 10 is greater than 5 AND less than 15.
Check if a word's length is greater than 3 OR less than 10.

Create a variable for your age. Create a boolean variable that is true if your age is 18 or over.

WHAT YOU LEARNED

You learned that `===` checks strict equality.
You learned that `!==` checks if things are not equal.
You learned greater than `>` and less than `<` comparisons.
You learned `>=` and `<=` for greater/less than or equal.
You learned to avoid loose equality `==` in favor of strict `===`.
You learned `&&` for AND (both true).
You learned `||` for OR (at least one true).
You learned `!` for NOT (flip the value).
You learned that comparisons produce boolean values.

The next chapter brings everything together with your first real programs.
BOOK 1: WHAT IS PROGRAMMING
Chapter 8: Your First Mini Programs

A GREETING PROGRAM

Let us build a program that creates personalized greetings.

Start with the data:

```
let firstName = "Alex"
let lastName = "Johnson"
let age = 28
let city = "Seattle"
```

Now create messages using this data:

```
let fullName = `${firstName} ${lastName}`
console.log(`Hello, ${fullName}!`)

let introduction = `My name is ${fullName}. I am ${age} years old
and I live in ${city}.`
console.log(introduction)

let nextYear = age + 1
console.log(`Next year, ${firstName} will be ${nextYear}.`)
```

Try changing the values at the top and running the code again. The messages update automatically because they are built from variables.

Add more features:

```
let nameLength = fullName.length
console.log(`Your full name has ${nameLength} characters.`)

let firstInitial = firstName[0]
let lastInitial = lastName[0]
let initials = `${firstInitial}.${lastInitial}.`
console.log(`Your initials are ${initials}`)
```

This program demonstrates variables, strings, template literals, math, and accessing string characters.

A SIMPLE CALCULATOR

Build a calculator that shows its work:

```
let num1 = 15
let num2 = 4

let sum = num1 + num2
let difference = num1 - num2
let product = num1 * num2
let quotient = num1 / num2
let remainder = num1 % num2

console.log(`Numbers: ${num1} and ${num2}`)
console.log(`Addition: ${num1} + ${num2} = ${sum}`)
console.log(`Subtraction: ${num1} - ${num2} = ${difference}`)
console.log(`Multiplication: ${num1} * ${num2} = ${product}`)
console.log(`Division: ${num1} / ${num2} = ${quotient}`)
console.log(`Remainder: ${num1} % ${num2} = ${remainder}`)
```

Change num1 and num2 to any numbers you want. The calculator handles all the operations.

Add some analysis:

```
let isNum1Bigger = num1 > num2
let isNum2Bigger = num2 > num1
let areEqual = num1 === num2

console.log(`Is ${num1} bigger than ${num2}? ${isNum1Bigger}`)
console.log(`Is ${num2} bigger than ${num1}? ${isNum2Bigger}`)
console.log(`Are they equal? ${areEqual}`)

let bothPositive = num1 > 0 && num2 > 0
let average = (num1 + num2) / 2
```

```
console.log(`Both positive? ${bothPositive}`)
console.log(`Average: ${average}`)
```

A TIP CALCULATOR

Real-world useful. Calculate tips at restaurants:

```
let billAmount = 85.50
let tipPercentage = 18
let numberOfPeople = 3

let tipAmount = billAmount * (tipPercentage / 100)
let totalAmount = billAmount + tipAmount
let perPerson = totalAmount / numberOfPeople

console.log(`Bill Amount: $${billAmount}`)
console.log(`Tip (${tipPercentage}%): $${tipAmount}`)
console.log(`Total: $${totalAmount}`)
console.log(`Split ${numberOfPeople} ways: $${perPerson} each`)
```

The numbers might have many decimal places. We can round them:

```
let tipAmountRounded = Math.round(tipAmount * 100) / 100
let totalAmountRounded = Math.round(totalAmount * 100) / 100
let perPersonRounded = Math.round(perPerson * 100) / 100

console.log(`Tip (${tipPercentage}%): $${tipAmountRounded}`)
console.log(`Total: $${totalAmountRounded}`)
console.log(`Per person: $${perPersonRounded}`)
```

The Math.round trick multiplies by 100, rounds to a whole number, then divides by 100. This gives two decimal places.

Add tip suggestions:

```
let tip15 = Math.round(billAmount * 0.15 * 100) / 100
let tip18 = Math.round(billAmount * 0.18 * 100) / 100
```

```
let tip20 = Math.round(billAmount * 0.20 * 100) / 100

console.log(`15% tip: $${tip15}`)
console.log(`18% tip: $${tip18}`)
console.log(`20% tip: $${tip20}`)
```

A TEMPERATURE CONVERTER

Convert between Fahrenheit and Celsius:

```
let fahrenheit = 98.6

let celsius = (fahrenheit - 32) * 5 / 9
let celsiusRounded = Math.round(celsius * 10) / 10

console.log(`${fahrenheit}°F = ${celsiusRounded}°C`)
```

Now the other direction:

```
let celsiusInput = 25

let fahrenheitOutput = (celsiusInput * 9 / 5) + 32
let fahrenheitRounded = Math.round(fahrenheitOutput * 10) / 10

console.log(`${celsiusInput}°C = ${fahrenheitRounded}°F`)
```

Add some context:

```
let isHot = fahrenheit > 85
let isCold = fahrenheit < 50
let isComfortable = fahrenheit >= 65 && fahrenheit <= 78

console.log(`Temperature: ${fahrenheit}°F`)
console.log(`Hot? ${isHot}`)
console.log(`Cold? ${isCold}`)
console.log(`Comfortable? ${isComfortable}`)
```

A TIME CONVERTER

Convert total seconds into hours, minutes, and seconds:

```
let totalSeconds = 9045

let hours = Math.floor(totalSeconds / 3600)
let remainingAfterHours = totalSeconds % 3600
let minutes = Math.floor(remainingAfterHours / 60)
let seconds = remainingAfterHours % 60

console.log(`${totalSeconds} seconds is:`)
console.log(`${hours} hours, ${minutes} minutes, ${seconds}
seconds`)
```

Math.floor rounds down to the nearest whole number. We use it because you cannot have 2.5 hours in this display.

Make it prettier:

```
let timeString = `${hours}h ${minutes}m ${seconds}s`
console.log(timeString)
```

Try different values for totalSeconds:

3661 seconds should be 1 hour, 1 minute, 1 second
7200 seconds should be 2 hours, 0 minutes, 0 seconds
45 seconds should be 0 hours, 0 minutes, 45 seconds

A WORD ANALYZER

Analyze a piece of text:

```
let text = "JavaScript is a powerful programming language"
```

```
let characterCount = text.length
let wordEstimate = Math.round(characterCount / 5)
let firstCharacter = text[0]
let lastCharacter = text[text.length - 1]

console.log(`Text: "${text}"`)
console.log(`Characters: ${characterCount}`)
console.log(`Estimated words: ${wordEstimate}`)
console.log(`Starts with: ${firstCharacter}`)
console.log(`Ends with: ${lastCharacter}`)

let hasUppercase = text !== text.toLowerCase()
let isLongText = characterCount > 100
let isShortText = characterCount < 20

console.log(`Has uppercase: ${hasUppercase}`)
console.log(`Long text (over 100 chars): ${isLongText}`)
console.log(`Short text (under 20 chars): ${isShortText}`)
```

We used text.toLowerCase() which converts all characters to lowercase. If the original text is different from its lowercase version, it must contain uppercase letters.

CHALLENGE PROGRAM

Combine everything you learned to build a mini profile card:

```
let userName = "Sarah_Dev"
let realName = "Sarah Martinez"
let followerCount = 1542
let followingCount = 328
let postCount = 89
let joinYear = 2019
let currentYear = 2024

console.log("=== PROFILE CARD ===")
```

```
console.log(`Username: @${userName}`)
console.log(`Name: ${realName}`)
console.log(``)
console.log(`Posts: ${postCount}`)
console.log(`Followers: ${followerCount}`)
console.log(`Following: ${followingCount}`)
console.log(``)

let yearsOnPlatform = currentYear - joinYear
console.log(`Member since: ${joinYear} (${yearsOnPlatform}
years)`)

let followerRatio = Math.round((followerCount / followingCount)
* 100) / 100
console.log(`Follower ratio: ${followerRatio}`)

let postsPerYear = Math.round(postCount / yearsOnPlatform)
console.log(`Average posts per year: ${postsPerYear}`)

let isPopular = followerCount > 1000
let isActive = postCount > 50
let isVeteran = yearsOnPlatform > 3

console.log(``)
console.log(`Popular (1000+ followers): ${isPopular}`)
console.log(`Active (50+ posts): ${isActive}`)
console.log(`Veteran (3+ years): ${isVeteran}`)
```

WHAT YOU BUILT

You built six working programs:

A greeting generator that personalizes messages
A calculator that performs and displays operations
A tip calculator for restaurant bills
A temperature converter between scales
A time converter from seconds to hours/minutes/seconds

A profile card with statistics

Each program combined multiple concepts:

Variables to store data
Different data types (numbers, strings, booleans)
Math operations
String manipulation
Template literals
Comparisons
Logical thinking

These are real programs. Simple, but real. You took data, processed it, and produced useful output.

WHAT COMES NEXT

Right now your programs run straight through from top to bottom. Every line executes once.

But real programs need to make decisions. If the user is logged in, show their dashboard. If not, show the login page. If the password is correct, let them in. If not, show an error.

Book 2 teaches you how to make decisions with if statements. Your programs will start to think.

After that, Book 3 teaches loops. Instead of running once, your code will repeat thousands of times. Process every item in a list. Count from 1 to a million. Search through data.

With decisions and loops, you can build almost anything.

You finished Book 1. You know the building blocks. Now we start constructing.